1992

NPSNET: Constructing a 3D Virtual World

Zyda, Michael J.

þÿ Z y d a , M i c h a e l J . , P r a t t , D a v i d R . , M o n a h a n , J a m e s G . a n d W i l s o n , K a l i n
þÿ C o n s t r u c t i n g a 3 D V i r t u a l W o r l d , i n C o m p u t e r G r a p h i c s , S p e c i a l I s s u e o
on Interactive 3D Graphics, MIT Media Laboratory, 29 March - 1 April 1992, pp. 147-156.
http://hdl.handle.net/10945/41592

NPSNET: Constructing A 3D Virtual World

Michael J. Zyda*, David R. Pratt, James G. Monahan, Kalin P. Wilson
Naval Postgraduate School
Department of Computer Science
Monterey, California 93943-5100
zyda@trouble.cs.nps.navy.mil
*contact author

Abstract

The development of 3D visual simulation systems on inexpensive, commercially available graphics workstations is occurring today and will be commonplace in the near future. Such systems are being constructed to move through and interact with 3D virtual worlds. There are a variety of goals for these systems, including training, planning, gaming and other purposes where the introduction of the physical player may be too hazardous, too expensive or too frivolous to be tolerated. We present one such system, NPSNET, a workstation-based, 3D visual simulator for virtual world exploration and experimentation.

Virtual World Systems

The attention to virtual world systems is particularly appealing to the researchers of the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School as our focus for years has been on the production of prototype 3D visual simulation systems on commercially available graphics workstations [9,18-25]. 3D visual simulation systems have many of the characteristics of virtual world systems in that their purpose has long been for visualizing and interacting with distant, expensive or hazardous environments. If we turn off some of our physical modeling, we can even simulate non-existent 3D environments, so we feel quite comfortable under the virtual worlds umbrella.

We do not study the construction of our 3D visual simulators on specially-designed graphics hardware. We instead assume that such hardware is available from commercial workstation manufacturers. We build 3D visual simulators on inexpensive graphics workstations instead of specially-designed hardware because of our observation that the performance numbers from the manufacturers are so suggestive.

NPSNET: Overview

The Graphics and Video Laboratory has been developing low-cost, three-dimensional visual simulation systems for the last six years on Silicon Graphics, Inc. IRIS workstations. The visual simulators developed include the FOG-M missile simulator, the VEH vehicle simulator, the airborne remotely operated device (AROD), the Moving Platform Simulator series (MPS-1, MPS-2 and MPS-3), the High Resolution Digital Terrain Model (HRDTM) system, the Forward Observer Simulator Trainer (FOST), the NPS Autonomous Underwater Vehicle simulator (NPSAUV), and the Command and Control Workstation of the Future system (CCWF).

Our current visual simulation efforts are on the NPSNET system, a workstation-based, 3D visual simulator that utilizes SIMNET databases and networking formats. The DARPA-sponsored SIMNET project had the goal of developing a low-cost tank simulator that provided a "70% solution" to the tank-war-gaming problem [17].

Unfortunately, the SIMNET system delivered has its graphics hardware and software suffering from a rigid specification based on 1983 graphics technology and was not designed to take advantage of ever faster and more capable graphics hardware and processor power. Low-cost for the project meant $250K per station. Instead, the contractor designed its own graphics platform, its own processing system, and wrote software that worked only on that platform. In NPSNET, we want to be somewhat more flexible BUT still interact with the DARPA investment.

The NPSNET system is an attempt to explore the SIMNET domain using a readily available graphics workstation, the Silicon Graphics, Inc. IRIS workstation in all its incarnations (Personal IRIS, GT, GTX, VGX...), instead of the contractor produced hardware. Our starting point is that we assume databases and network packet formats in a form similar to those utilized by the actual SIMNET system but allow the flexibility for continuing evolutions in efficiency.

NPSNET is a real-time, 3D visual simulation system capable of displaying vehicle movement over the ground or in the air. Displays show on-ground cultural features such as roads, buildings, soil types and elevations. The user can select any one of 500 active vehicles via mouse selection and control it with a six degree of freedom spaceball or button/dialbox. In between updating events, all vehicles are dead reckoned to determine their current positions. Speed in three dimensions and the location of the vehicle can accurately be predicted as long as the speed or direction of the vehicle does not change. Vehicles can be controlled by a prewritten script, or can be driven interactively from other workstations, as the system is networked via Ethernet. Additionally, autonomous players can be introduced into the system via a programmable network "harness" process (NPSNET-HARNESS).

As obvious from the above overview, NPSNET is in many ways a departure from the goals of SIMNET. We can "push the envelope" of real-time, workstation-based virtual reality while providing *a workstation-based SIMNET node*. We present our plan for the overall NPSNET effort in the following sections to provide an understanding of what is required to construct such a system.

SIMNET Database Display Work

The first effort in any virtual world development is obtaining the data that represents the world to be modeled. For 3D visual simulations, this usually begins with a large 2D grid of elevation data that is turned into a 3D terrain carpet.

Once the terrain carpet has been extracted and displayed, attention then turns to on-ground cultural features and 3D vehicle *icons*. On-ground cultural features include roads, forest canopies, trees, building, corrals and other stationary objects. Many cultural features are provided in 2D and have to be projected onto the terrain. Significant work must be done to accomplish this. There is the preprocessing work to turn 2D linear features like roads into 3D, correctly projected onto the terrain carpet. Projecting planar 3D road segments onto the terrain carpet is also not easy. The problem is that it requires projecting the road polygons onto the same place as the terrain carpet. Under z-buffering, the standard hidden surface elimination method for graphics workstations, coplanar, coincident polygons cause what is known as z-buffer *tearing* [1]. We see scan lines alternately colored with the underlying terrain color and the road color. We solve this by drawing the underlying terrain polygon first into the RGB planes with z-buffering on but modifications to the z-buffer off. We then draw the road overlay. Modifications to the RGB planes are then turned off and the underlying terrain polygon is again drawn, this time with modifications to the z-buffer on. This procedure must be done for all coplanar features in the system. It requires that underlying layers be drawn multiple times and in an ordered fashion. The visual simulator must handle this in a general fashion. It is just part of the complexity of building such systems.

3D vehicle *icons* are the next consideration in constructing our virtual world system. We call them 3D icons in that the goal is not realism but rather low resolution indicators of players on the terrain. Low resolution means whatever level of detail the user of the final system is willing to live with.

Hierarchical Data Structures for Real-Time Display Generation

If the modeled world is simple, just blasting all the polygons through the graphics pipeline ought to get satisfactory display results. Since NPSNET uses data from the SIMNET Database Interchange Specification (SDIS) for an actual 50km x 50km terrain area of Fort Hunter-Liggett, California and has a resolution of one data point for every 125 meters [6], this will not do.

Hierarchical data structures are the heart of any complex real-time, 3D visual simulator. Such data structures, in conjunction with viewing information, provide for the rapid culling of polygons comprising the terrain carpet, the cultural features, the 3D icons and any other displayable objects. The purpose of this operation is to minimize or reduce the flow of polygons through the graphics pipeline of the workstation's hardware. A classic reference to understand this problem in more detail is [2]. The culling operation is performed through the traversal of a data structure that spatially partitions the displayable data. The appropriate hierarchical data structure to use is problem domain dependent. As we have adopted NPSNET to additional tasks, we have had to modify and change our data structure.

Expanding the Terrain Area

In order to increase performance, the initial NPSNET dataset has been divided into 2500 text files based on the one kilometer standard of the military "grid square" with each file containing data for one square kilometer. These were preprocessed into binary format and three additional lower resolutions generated (250, 500 and 1000 meter), together with fill polygons for each level. The final form of the dataset is 2500 binary files, each containing a multiple-resolution (4 level) description of the terrain for one square km, stored as a heap-sorted quadtree [7,12].

The final format for the binary terrain data files is designed for fast access using the C function *fread()*. All polygon descriptions are stored in memory-image format, therefore, no data conversion has to be done during paging. The 2500 files resulting from preprocessing contain:

- Count of polygons in each node of full four level quadtree (85 total).
- Total polygon descriptions in the file.
- Multi-resolution description of terrain in this square kilometer stored in quadtree heap-sort order, lower resolutions first (Figure 1).

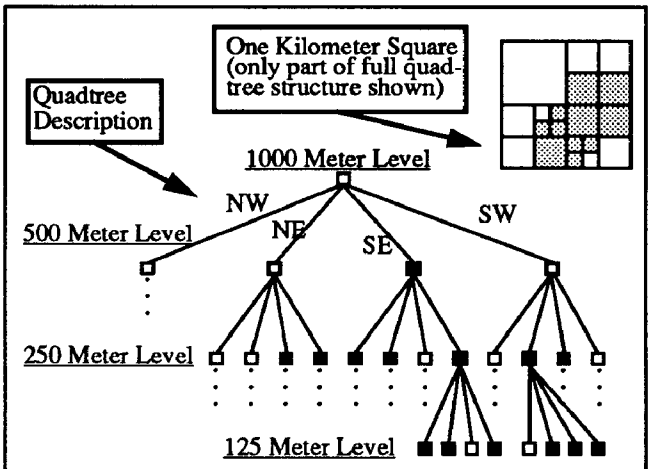


Figure 1 - Multiple Resolution Quadtree

As the final dataset is too large to store in main memory at one time, and we do not wish to limit the simulation to some smaller area, paging terrain data through a dynamic algorithm is required.

A 16km x 16km active area was chosen based on considerations for memory size of available workstations, frame rates, required field of view and desired range of views. This amount of terrain data is in main memory at any given time and available for rendering. Sixteen kilometers allows a seven kilometer field of view in all directions for immediate rendering with one kilometer acting as a buffer to ensure terrain is fully paged in before attempting to render it (Figure 2).

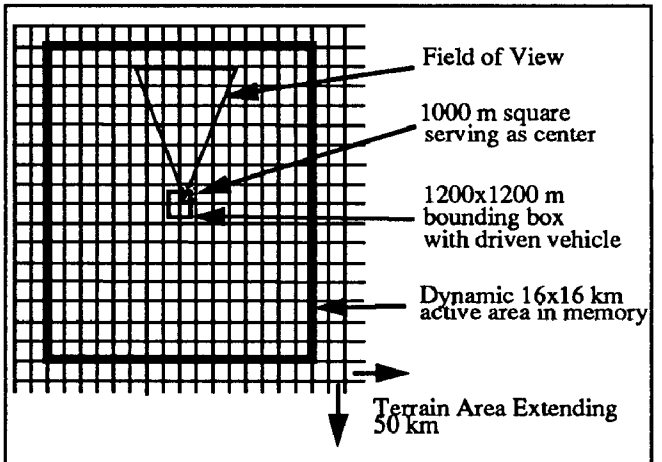


Figure 2 Active Area of Terrain in NPSNET

On multi-processor workstations, the simulator does not wait for additional terrain to be paged in. Instead, the additional CPUs are used to page in the terrain in parallel.

Terrain Paging Algorithm

When the simulator is initialized, the driven vehicle is centered on a 16 x 16 active area. The indices of the center one kilometer square containing the driven vehicle become the notional center. Data is loaded into the appropriate elements of a 50 x 50 array, and a bounding box is established around the driven vehicle (centered on the index of the center square). When the driven vehicle reaches the bounding box in any direction, memory space is freed in the direction opposite of travel, terrain is paged in the direction of travel, and the bounding box moves.The size of the bounding box can be adjusted as required by vehicle speed/turn rate characteristics. Terrain paging is independent of the hierarchical data structure implemented.

Terrain Rendering

Terrain rendering involves several steps in NPSNET:

- ❑ Determine which 1000m x 1000m squares are actually in the field of view.
- ❑ Determine resolution within each 1000m x 1000m square (there may be at most two resolutions), including which fill polygons are needed.
- ❑ Render the terrain.

Two algorithms are involved. One checks to see if a polygon is within the field of view by calling a procedure that checks for the intersection of a point (each point of the polygon) and a polygon (the triangle composing the field of view) [3]. The other determines the resolution, essentially which nodes of the quadtree to render, by checking the intersection of nodes with concentric circles corresponding to ranges of the resolutions [13]. The circle-rectangle and point-polygon intersection algorithms are applied repetitively to render only terrain within the field of view and at the appropriate resolution levels. Figure 3 depicts multi-resolution within the field of view. NPSNET is graphics bound. Therefore, the computational expense of the above algorithms is better than rendering terrain not actually in the field of view.

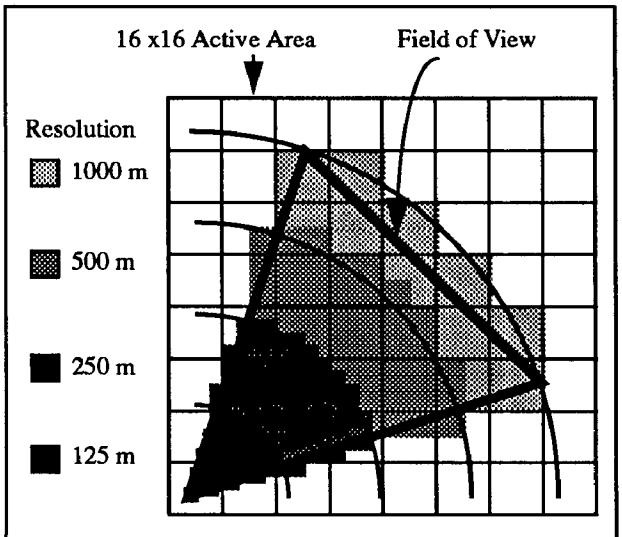


Figure 3 - Multi-Resolution Rendering

Implementation of the above has resulted in a doubling of the performance of the simulation over high resolution rendering alone. However, performance when large numbers of objects (trees, vehicles) are present in the viewing area does not change.

NPSOFF: Overview

The development of interesting virtual world systems requires the modeling of many different graphical objects. How these objects are represented in the system plays a major part in determining the capabilities and efficiency of the system. We use a simple, flexible object description language to model graphical and some non-graphical aspects of our objects called NPSOFF.

NPSOFF is a language system that consists of "tokens" that represent graphical concepts. These tokens are combined in an ASCII file to represent an object. The object can then be referenced by an application in an abstract manner. The application does not need to know the details of how the object is composed. The level of abstraction that NPSOFF provides offers numerous advantages that are discussed below. NPSOFF objects can have varying levels of complexity to represent a wide range of graphical objects and environments. NPSOFF also serves as a standard for application development. This makes general purpose tools plausible and extremely useful.

Functional Description

The NPSOFF language can be broken down into tokens. In early versions of the language, the tokens corresponded almost one for one to GL functions. Later versions have added more abstraction and flexibility. The language tokens simplify the interface to the GL library by labeling components and help encapsulate some of its complexity.

NPSOFF extends the GL interface by allowing many system settings to be named. Naming system definitions allows us to build libraries of commonly used settings like materials and textures.

NPSOFF tokens generally belong to one of three categories: definition, display or characteristics/composition. The definition tokens define graphics system settings. Definition tokens define lights (normal and spot), lighting models (normal and two-sided), materials, textures, and colors. Definition tokens are named and stored in tables for later access.

Display or execution tokens make up the bulk of NPSOFF. Display tokens represent a change in graphics system state or graphics primitives. They are stored in a sequential display list in the order that they appear in an NPSOFF file. Example tokens that change the system state are: ***setmaterial***, ***setlight***, ***settexture***, etc. Each of these tokens has a name argument that corresponds with an earlier definition. In the case of ***setlight***, the named light is associated with one of seven possible light numbers [14]. These state tokens make it easy to manipulate the graphics pipeline. Complex lighting and shading effects can be done with NPSOFF in a simple and straightforward way.

The graphics primitives used in NPSOFF are: polygons, surface (polygon with vertex normals), triangular mesh and lines. Additional display tokens perform manipulations of the system matrix stack. NPSOFF objects and components of objects can be transformed within the object definition file. The tokens ***loadmatrix***, ***multmatrix***, ***pushmatrix***, ***popmatrix***, ***rotate***, ***scale*** and ***translate*** define stack manipulations.

The third category of NPSOFF tokens allow the user to define object characteristics and composition. This allows a high level of abstraction and supports complex graphics techniques. Two of the main abstractions are composite objects and polygon decaling. NP-

SOFF objects can be named and contain nested object definitions. The nested definitions can contain any display tokens. This structure allows multiple related objects to be treated as a single object for graphics display. It also minimizes the duplication of primitive definitions. Objects are defined with the ***defobject*** token and displayed with the ***callobject*** token. This structure is flexible and useful for building complex objects from simpler sub-objects.

Using an NPSOFF object is also simple. Essentially the user needs to use only three function calls to access and display an object. There are many more programmatic entrypoints to NPSOFF but many of them deal with in-memory manipulation that is not needed for standard use. They are used primarily by tools that build or manipulate NPSOFF objects.

Physical Modeling Support

In the past, simulations developed in the Graphics and Video Laboratory have each handled physically-based modelling (PBM) independently and internally. The latest extension of the NPSOFF system is an object-oriented PBM system [8]. These enhancements give NPSOFF objects physical characteristics and provide mechanisms to control an object's motion given a list of internal and external forces on the object. Objects are handled in an enclosed reference called the "environment". All objects that participate in the NPSOFF PBM system are members of the environment.

The NPSOFF PBM system models object rigid-body dynamics using a Newtonian framework. An object can be given many physical properties using the ***defphysics*** token. These properties include the object's initial location and location constraints in the environment, initial orientation and orientation constraints, initial linear and angular velocities and constraints on each, the object's mass and center of mass, the object's ability to absorb forces (elasticity), the dimensions of a bounding volume and a local viewpoint for the object. Each object can also use its own system of measurement. The ***defunits*** token allows the user to specify the units of measurement for dimensions, force magnitude and mass. This capability was incorporated to accommodate the use of object models from various sources. The PBM system uses reasonable constant or calculated defaults for all physical characteristics so none of the properties is required to be present when object physical characteristics are defined.

Forces are defined and added to an object's force list with the ***defforce*** token. Two types of forces are supported: deforming and non-deforming. Deforming forces are used for object explosions and bending. Non-deforming forces are used to alter an objects linear and angular velocities. Forces can be specified as awake or asleep. This allows the selective application of previously defined forces. The characteristics of a force defined with ***defforce*** are: type (deforming/non-deforming), origin relative to object center and origin constraints, force direction vector, magnitude and magnitude constraints and force state (asleep/awake).

The run-time interface of the NPSOFF PBM system is simple and flexible. Once the PBM environment has been initialized, the user can add or delete objects from the environment, add and modify global forces, modify object physical characteristics, add and modify object force characteristics and modify object and force states. The environment is processed once each display cycle. The processing involves resolving forces, calculating object states and displaying the objects. The NPSOFF PBM system provides us with a simple environment to model object dynamics and interaction. This is one of our first steps to add more physical reality to our applications.

Advantages

NPSOFF provides many advantages to the researchers in the NPS Graphics and Video Laboratory:

- NPSOFF allows an application independent description of graphical objects. Objects can be designed and maintained by general purpose tools. Collections of objects can be built and shared with other researchers.
- NPSOFF adds a level of abstraction that greatly simplifies application development. Also, by having a large collection of common objects, developers can concentrate on how objects should be used rather than designing and rendering the objects.
- NPSOFF provides a simple, object oriented, run-time interface to an object. Functions such as read_object(), display_object() and delete_object() all operate on individual objects in memory. Many functions are provided so flexible manipulations are possible.
- The stand-alone, reusable nature of NPSOFF objects encourages the use of common libraries of definition tokens such as materials and textures.

Support Tools

The wide use of NPSOFF in our laboratory has led to a variety of tools to aid in the design and maintenance of NPSOFF objects. These tools include: The OFF calculator, NPSME - a material editor, NPSTE - a texture editor, NPSICON - a model builder and NPSMOVER - a physically based design editor.

The OFF calculator allows in memory manipulation of NPSOFF objects using a simple command line interface. Using the OFF calculator, objects can be transformed (transformation applied to all primitives), primitives can be added to an object, graphical objects (spheres, boxes, etc.) can be added to an object and objects can be concatenated.

NPSME is a material editor that helps manage libraries of materials [26]. It reads and writes material definitions. Material definitions can be selected from the library for viewing and editing. The material editor helps us to maintain a large collection of material definitions used by NPSOFF objects in our applications. The ability to interactively design and modify material definitions is very important to rapid application development.

NPSTE is a texture editor that helps manage libraries of NPSOFF texture definitions [26]. NPSTE can use images in many formats as textures. Portions of an image can be copied and used as a texture image. Textures can be viewed on any NPSOFF object using either the texture coordinates specified in the object or automatically generated coordinates using the GL function *texgen()* [14]. Textures can be edited using a simple pixel editor. Finally, a texture definition can be saved in a library of textures and the library saved as an NPSOFF file. The texture editor lets developers interactively create, select and view textures independent of a developing application.

NPSICON is an interactive object design tool [10]. NPSICON lets a developer design or modify NPSOFF objects using a set of predefined building blocks. NPSICON is designed to be used primarily to build vehicular models. Objects can be edited and transformed in many ways and then saved to an NPSOFF file. NPSICON allows rapid prototyping of vehicular objects for use in applications. It also allows developers to modify existing models quickly and easily.

NPSMOVER provides an environment for users to design and test physical dynamics of NPSOFF objects [8]. NPSMOVER reads any NPSOFF file and assigns default physical characteristics if not present. The user can then adjust all physical characteristics of the object. Forces can be defined and added to an object's force list using interactive controls. Once the object's initial conditions, constraints and characteristics are set and the forces acting on the object

are specified, the dynamics can be "turned on". The user can observe the effects of the forces and make necessary adjustments. Once the user is satisfied, the object can be saved to an NPSOFF file with all the needed tokens. The NPSMOVER tool provides a simple, interactive environment to view and adjust an object's basic dynamic behavior.

NPSOFF Future Directions

Current and future projects at NPS are working to extend and improve NPSOFF including support for defining inter-object relationships and constraints. This would allow the composite object structure to be extended to where each subobject has physical properties and affects the behavior of the whole object. Also the notion of linked objects will be explored in the context of NPSOFF. This will allow the realistic modeling of such things as vehicle controls (e.g. aircraft stick movement changes control surface which changes forces on whole aircraft).

Another area that future research will address is animation support within NPSOFF. Support for continuously animated portions of an object (vehicle antennae) or constraint management of subobjects (doors, arms, etc.) would be very useful to our researchers. Such a system would benefit from the standardization that NPSOFF provides and offer much more capabilities to developers.

The NPSOFF system is object-oriented in its design and use but is implemented in a non-object oriented language. Modifying or extending the current system is time consuming and error prone. We are currently redesigning NPSOFF to be truly object-oriented and implementing it in C++. The main benefits of moving to an object-oriented implementation will be increased extensibility through inheritance and polymorphism and better maintainability.

Collision Detection

In earlier versions, NPSNET did not detect nor respond to vehicle collisions. Without collision detection and response, the realism was poor. Even with texturing, environmental effects and realistic looking vehicles, the virtual world falls apart the first time one vehicle drives through another. A possible solution to this problem would be to prevent interpenetrations by bouncing objects off of each other after any contact, but this is rarely accurate. Another possible solution is to destroy the objects involved in collisions. A third option is to combine these two solutions along with varying stages of damage to involved objects depending upon the physical characteristics of the involved objects. The current version of NPSNET detects and responds to collisions between objects in real-time. Detection is sufficiently fast to allow the time needed to respond properly. Response time is dependent upon the level of physically-based modeling.

Collisions with Fixed Objects

The algorithm for collisions with fixed objects constantly checks moving vehicles to determine if a collision has occurred. The position of the moving vehicle is updated constantly. Consequently, as soon as a vehicle is moved and its position is updated, it is checked for a collision. In order to maintain a real-time speed, the scope of the collision detection is severely limited. A collision with fixed objects is checked only if the moving vehicle is below a threshold elevation. All fixed objects are in some way attached to the terrain and thus below that threshold elevation. If an object is below that elevation, NPSNET runs through a linked list of fixed objects which are attached to the current gridsquare.

Collisions with Moving Objects

A collision with other moving objects is more complicated since any other moving vehicle or object has the potential for colliding with the vehicle we are checking. The potential exists for checking up to 500 vehicles and any of their expendable weapons. Consequently, the scope of the collision detection range has been limited in several ways.

As soon as each vehicle is moved, its position is checked against the position of the neighboring vehicles. If the X or Z position of any other vehicle is within 100 meters of the checked vehicle then those two vehicles are sent to the second level check. At the second level check, the distance between the two vehicles is calculated. If this distance is less than the combined radii of the two vehicles, then a collision has occurred and the third level collision check is done. A rudimentary form of ray tracing determines the actual point of collision.

If worst case numbers are used to determine the implicit range limitations of all vehicles, it can be shown why this culling is fairly accurate. Reasonable speed limitations of the various types of vehicles are used to calculate worst cases for each (Table 1). Consequently, the movement across more than two gridsquares within one tenth of a second, one frame, is unlikely.

Table 1: VEHICLE MOVEMENT LIMITATIONS

	KPH	m/sec	Frames/sec	m/Frame
Land	60	16.6	10	1.66
Sea	50	13.8	10	1.38
Air	1000	277.7	10	27.7

Collision detection is accomplished by determining if one object's bounding sphere has interpenetrated another. The radius used in the spherical check is the maximum distance from the center of the object to the furthest outer vertex. In the collision response portion of the system, the actual object's penetration point is determined. A slightly smaller value than the actual radius of the object is used for the radius. This produces a more realistic collision possibility since it increases the likelihood of an actual collision of the checked objects and not just their spheres. Once the collision has been detected, the extent of damage and collision response are determined.

Collision Response

Collision response is handled by a function which takes into account speed and angle of impact, mass of the objects involved, explosive potential, resistance to destruction, moldability of the objects, rigidity and fabricated spring forces which determine the bouncing-off effect and likelihood of survivability. Each of these factors is weighted in order to provide as realistic an effect as possible while maintaining the environment in real-time.

Moving Objects

In the case where two moving objects impact, all of the physically-based modeling characteristics of each object must be considered. The collision point must be known to create realistic responses in the involved objects. The collision point determines the point for any type of bending, crumpling and molding. Moreover, if the point of collision is part of a wall that is interconnected to several other walls then there will have to be corresponding responses in those interconnected walls. The only way to find the collision point is through ray tracing.

The first ray is shot from the center of a moving object towards the center of an adjacent object to determine a possible point of collision. This collision may simply be between the bounding spheres of the two objects and not the actual objects themselves. The intersection between the first ray and the second object's bounding sphere is used to specify the direction of a second ray originating from the adjacent object's center.

The second ray determines if one of the object's actual polygons was penetrated. This second ray is the ray used in Haines' algorithm. This algorithm from Glassner [4] was adapted for use in the collision point determination. It involves running through the list of polygons that comprise the adjacent object and determining if the second ray intersects the plane containing the polygon. If no intersection is found once all of the polygons have been checked, then only the spheres were penetrated and not the objects themselves.

Reactions

The proper response is performed by comparing the characteristics of two objects involved in the collision. For fixed objects, the responses include several degrees of damage, based upon the speed and mass of the colliding object. Up to three levels of damage plus the original undamaged fixed object are available for display after a collision. For mobile objects, the response depends upon the angle of impact as well as the speed and mass of the two involved objects. The mobile object reacts by either bouncing away or being destroyed and exploding. In the special case of contact by munitions, the only response is an explosion.The limited number of options available for the response to the collision keep the response fast to maintain the real-time criteria. The collision point and direction of travel are passed to another module that handles physically-based modeling of object movement. This function's implementation can be seen in [8].

SIMNET Networking Integration

SIMNET networking integration is part of our NPSNET efforts on software structures for world modeling in that networking provides the locations and actions of other players in our visual simulators. We use Ethernet and TCP/IP multicast packets of our own design for the current NPSNET system. We are in the process of integrating the networking system with the SIMNET standard packets as the full description and documentation is now available. This connection to SIMNET will provide players, weapons firing and other state information with which we can test our world modeling efforts. At a later stage, we hope to examine some of the available work on higher speed networks, such as FDDI, as it becomes commercially available and relevant.

NPSNET-HARNESS Structure

The NPSNET-HARNESS process was developed to allow the rapid integration of different components into the NPSNET simulation system and in partial response to Ethernet's speed and addressing limitations [15]. The high level structure of the network harness is shown in Figure 4. The harness is divided into two main sections, the Network Daemon and the User Program Interface, which communicate via shared memory. The principle purpose of the Network Daemon is to provide low level data and network management support for user written NPSNET "player" programs. Player programs developed by users are stand alone applications that provide specific world interaction functionality.

The User Program Interface consists of a set of routines that allow the programmer to interact with the network at a higher level of abstraction. These functions include setting up the shared memory space with the network daemon, creation of a network read key, message formatting, and the actual reading and writing of network messages.

Message Types

One of the interesting things about the Ethernet network is that it is more efficient to have a few long messages rather then many short messages[16]. This influenced the creation of five message types and formats.

The message types, NEWSTATMESS and DELSTATMESS, are used when a a station enters the network and when it no longer is an active player in the networked environment. These are used solely as administrative messages and do not affect the appearance of any vehicle.

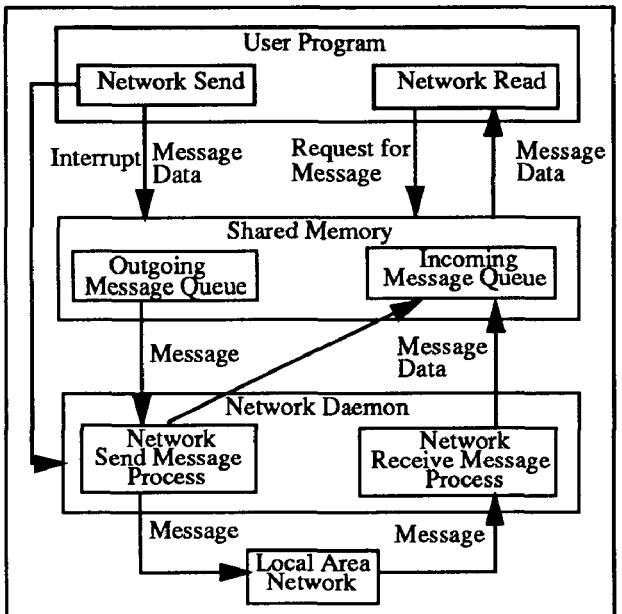


Figure 4 Structure of the NPSNET Network Harness

One of the features of NPSNET is the capability of allowing the user to change vehicles during the execution of the simulation. The SWITCHMESS notifies all the other nodes on the network that the user has changed vehicles. This does not affect the appearance of any of the vehicles.

The UPDATEMESS is the largest message used in NPSNET and it is also the most common, accounting for almost all the network traffic. Before we discuss this message, the concept of the state of the vehicle must be covered. As mentioned previously, the vehicle's position is updated only after a speed or direction change. The tilt and roll of the vehicle can be derived from the location on the terrain and need not be sent across the network. Additionally, the orientation of the turret, the gun elevation, vehicle destruction, and weapons firing all change the state of the vehicle. Whenever any of these state parameters change, a message must be sent to update the other network nodes.

Since it is more efficient to have a few long messages rather then many short ones, we combined all of the vehicle state parameters into a single message. This has the additional benefit of updating all of the vehicle parameters at the same time to ensure accurate placement and orientation of the vehicle.

NPSNET-HARNESS Future Directions

Currently there are two major efforts underway concerning NPSNET-HARNESS. The first of these is the porting of the system to Sun SPARC workstations. We envision providing the user a standard network interface for both the IRIS and Sun workstations. This will allow the development of Autonomous Agents (AA) and Semi-Automated Forces (SAF) that can interact with the vehicles that are driven on the IRIS workstations. Our 100+ departmental Sun workstations would then serve as a distributed multiprocessor.

The second major effort is the utilization of the SIMNET Protocols [11]. As shown in Figure 5, we plan on constructing an interface between the User Program and the Network Daemon to convert the format of the protocols between the internal and external protocol. This will later be extended to the DIS Protocols [5] as well. The use of a translator will isolate the programmer from changes in the protocols. Naturally, we will increase the number of messages available to the user when we use the new protocols, but the old message formats will remain.

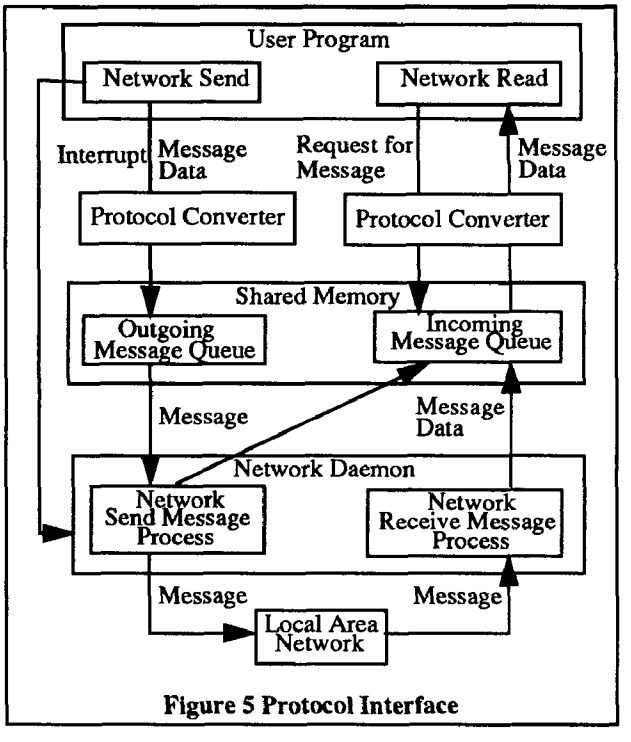


Figure 5 Protocol Interface

Semi-Automated Forces

The current DARPA SIMNET system has a semi-automated forces (SAF) component in it. The SAF system provides autonomous players to SIMNET when sufficient numbers of actual, interactive players are not available or affordable. The Graphics and Video Laboratory has considerable experience in generating such players as our visual simulation efforts have a close coupling to our department's artificial intelligence and robotics efforts [20,21]. We are continuing those efforts and expanding that work to take advantage of the available parallel processing capabilities of our workstations.

NPSNET-MES: Overview

Earlier versions of NPSNET used randomly guided vehicles to populate the battlefield. These vehicles had very little intelligence and were only capable of firing back at an attacker or running away. It is not enough to have random vehicles moving about the battlefield without a mission; we must populate the battlefield with combat formations that act semi-autonomously as well. The NPSNET Mobility Expert System (NPSNET-MES) provides realistic semi-automated forces (SAF) to introduce sufficient numbers of unmanned players into the system to make the simulation more challenging and exciting. NPSNET-MES consists of two components: a path generation module and a vehicle controller module. The path generation module determines the SAF route and mission based upon the SAF controller input. The vehicle controller module uses the programmable harness, NPSNET-HARNESS, to multicast data packets via Ethernet to control the SAF vehicles during the simulation. NPSNET-MES integrates SAF into an already existing network simulator such that no changes are necessary to NPSNET.

Problem Description

One of the major objectives of our work is to determine the best approach to integrate semi-automated forces into an already existing simulation. The following are the minimum capabilities of the semi-automated forces: The SAF controller specifies a path that includes start and goal points with possible way points along the route. The SAF must negotiate all known obstacles without hitting them in a relatively optimal path. The SAF vehicles within a SAF formation must follow the lead SAF vehicle such that they maintain relative positions and do not collide with each other. The SAF controller specifies the number of combat formations as well as the number of vehicles, speed and type of each combat formation. When a SAF vehicle is killed, it no longer moves. NPSNET-MES integrates the SAF into the existing NPSNET without any change to the system. Once the SAF controller determines the SAF prerequisite information, NPSNET-MES makes that information available to NPSNET for use during the simulation. These basic considerations drive the requirements for the NPSNET-MES prototype system.

Integration with NPSNET

To get the desired results, NPSNET-MES is designed to act in a stand alone mode. This means that NPSNET-MES integrates the SAF into NPSNET by using the existing set of programmable network harness routines, NPSNET-HARNESS. The main problem separates into two distinct subsets: designing semi-automated forces that can navigate and travel a specified path and transmitting the information generated by the first part.

Path Generation Module

This module is a 2D map/interface that the SAF controller uses to perform SAF vehicle placement and route selection. The SAF controller is able to control the SAF parameters, such as number of SAF formations, number of vehicles in each SAF formation, and type of SAF vehicles and input a desired path with intermediate rendezvous points as well as a speed for each path segment.

NPSNET-MES stores this information in a file available to the vehicle controller module.The path selection criteria for this module is not an optimal path, rather it is a relatively simple path that is found quickly. This module generates a path based on *a priori* obstacle information using a circle world.

Computational versus *a priori* Path Planning

The path generation module's path generation algorithm uses a modified breadth-first search of a bounding box rather than the more traditional artificial intelligence approach of *a priori* generated paths because it is more efficient and less complex. The compu-

tational approach searches the bounding box, shooting a line between the start and goal to determine if the goal is visible from the start. If the path has an obstacle, then the path finder is called recursively until a path is found around obstacles enroute to the goal. The NPSNET-MES path generation module algorithm bounds the search area using the start and goal points to limit the search within a box.

An *a priori* path generation produces paths for the entire database requiring a longer amount of time and more memory to store those paths for quick access than a computational generation. The recursive path planner grows in a linear fashion versus a non-linear growth for the more traditional *a priori* method (Figure 6).

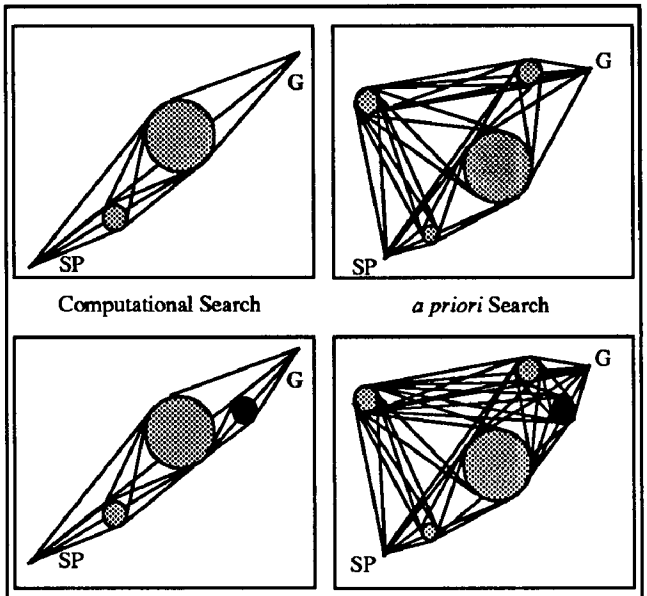


Figure 6 Computational vs. *a priori* Search

Path Generation Module

The path generation module is the interface for NPSNET-MES with NPSNET. This program places the generated paths in a sorted linked list by ascending order of time. A path point time is a running total time for the vehicle from the start up to that point. Using the system clock to maintain relative time, the paths are taken off a priority list. The NPSNET-HARNESS sends updated messages reflecting the new vehicle position, direction, and speed to NPSNET. NPSNET receives the path data and the SAF vehicles respond to the vehicle controller commands ensuring that the SAF vehicles stay on track with the generated paths.

NPSNET-MES Results

NPSNET-MES provides a relatively efficient solution to finding a good path for the SAF vehicles. The path found by the path generation module does not attempt to find the best solution only a good solution, since the human that it emulates usually only finds a good solution when conducting path planning. The vehicle control module provides the necessary interface between NPSNET-MES and NPSNET so that the SAF forces travel as they would in real life.The system provides a realistic friend or foe force on the simulation battlefield. NPSNET-MES effectively integrates SAF into NPSNET. This system is a prototype for research, therefore it has many potential capabilities that can be added at a later time.

Path Generation Module Limitations

- ❒ No dynamic path planning for the SAFs to react to other players during the simulation.
- ❒ Produces only one combat formation type for the entire mission.
- ❒ Terrain slope considerations are not incorporated in the path planning algorithm.

The most serious limitation with the system is the inability of the SAF to react to other players in the simulation. The SAF missions are pre-set before the simulation begins and cannot be altered once it commences. This was a design decision made at the outset of the project. The deficiency can be corrected by incorporating a local path generation capability within the vehicle controller module. When a SAF comes within range of an active player, the vehicle controller module path generation function would generate a local path around the moving obstacle and then the SAF reenters the previous path at the closest point.The path generation module places all follow-on vehicles in a column of wedges. This is a good movement formation, but there are many occasions where other formations would be appropriate. This additional flexibility is possible by giving the SAF controller some options during his path planning preparation. Terrain slope considerations are not incorporated into the path generation module because the design calls for a fast and efficient path planner. Terrain analysis requires more computation per path segment since the path generator evaluates each path segment terrain slope for terrain selection.

Path Generation Module Limitations

- ❒ Limited SAF vehicle reaction to active simulation players.
- ❒ Projected and actual path plots deviate due to clock speed and network transmission times.

A design decision was made early in the design phase rejecting multiple reaction capabilities. The SAF vehicles die when attacked because NPSNET-MES no longer sends update positions and reduces the speed to zero. By increasing the number of items that the vehicle controller module checks from the network, the reaction capability is upgradeable.

The final limitation is not a serious one since deviations are small and the shifting movement is not conspicuous. To fix the problem, the system must be able to operate at the millisecond rate or faster since the path points are in an ascending order queue. Some path points may have the same time stamp causing a delay for at least one of the SAF vehicles. NPSNET-HARNESS is not able to operate faster than its current rate due to hardware system limitations. The limitations create a bottleneck because there is only a single wire and single port on the Ethernet. There will always be some error due to transmission time delay, but this effect is negligible as long as the machines are in relative proximity.

Aural Cues for 3D Visual Simulation

A realistic virtual world must include aural cues about the objects in the world. These cues should provide feedback about the user's environment and actions taking place. A recent addition to NPSNET is the support of sound feedback to the user.

The addition of sound to a complex virtual world is itself complex. Often, parallel event generated sounds are routed to sound devices which are serial in nature. This imposes a severe limitation that must be worked around.

One solution we are investigating involves a process that can intelligently manage requests for sound issued from NPSNET.

This process would have several responsibilities:

- ❑ Receive sound requests, resolve multiple similar sounds into a single sound that can represent them and throw away requests of significant age.
- ❑ Coordinate requests for continuous sounds (e.g. background noise, other vehicular noise, etc.).
- ❑ Manage the use of multiple sound production devices (e.g. samplers, keyboards, MIDI devices, etc.).
- ❑ Facilitate the use of 3D sound.

This sound manager process would allow NPSNET to deal with sounds in a fairly abstract manner. Only knowledge of classes of sounds would need to be shared between NPSNET and the sound manager. This will allow us to modify the sound manager easily without affecting NPSNET.

Currently, sound support in NPSNET is limited. We use a Macintosh IIci running in-house software to play digitized sound files. The Macintosh is connected to an IRIS workstation running NPSNET by a serial link between RS-232 ports. When NPSNET wants to produce a sound, it issues a request for a specific sound to be played by the Mac via the serial port. The Macintosh queues the request, locates and plays the sound in the system resource. There are several limitations to this solution:

- ❑ NPSNET must know specific sound names that exist on the Macintosh and request them by name.
- ❑ Currently all sound files on the Macintosh must reside in the system folder. This limits the number of sounds that are available.
- ❑ Only discrete sounds are currently used. There is no notion of continuous sounds.
- ❑ A single device with one channel is used to reproduce the sound. This can lead to a backlog of requested sounds.
- ❑ The queue of sound requests on the Macintosh can become overloaded due to the above backlog. This can result in lost sounds, delayed sounds or queue overflow.

Ongoing work with sound and NPSNET is approaching the model outlined above. We are beginning to investigate high quality sound samplers and MIDI devices attached to the Macintosh to collect, create and reproduce various sounds. Sophisticated sound editing, sequencing and control software on the Mac give us many options for creatively employing aural feedback in NPSNET. Support for 3D sound is also under research.

Since many sounds are object-based, NPSOFF objects will support the description and management of sound that pertain to themselves. The sound control with NPSOFF will provide a standard use of sounds and facilitate the collection of sound definitions just as we collect materials and textures.

We believe that sound is an integral part of any serious virtual world simulation. We are actively pursuing efficient, extensible and effective solutions to integrating sound into NPSNET.

NPSNET: Current Performance

The current NPSNET system runs on a variety of platforms. Our highest performance system in the laboratory is the Silicon Graphics, Inc. IRIS 240 VGX with 64MB CPU memory. The VGX system is listed by the manufacturer as being capable of some 1 million triangles per second, z-buffered and Gouraud-shaded. On that system with terrain texturing on, NPSNET shows 6 frames/second with many objects in the display and 9 frames/second with few visible objects. The system has a switch to turn off texturing of the terrain and the frame rate roughly doubles respectively.

The performance of NPSNET is not affected by the addition of the collision detection and response modules as it is. The response time for detection of fixed objects is adequate regardless of the speed of the moving objects. However, for collisions between two high speed objects, collision detection is sometimes slow.

Fully Interactive and Detailed Virtual Worlds

While the NPSNET virtual world is not yet complete (and may never be), it is still a consequential and somewhat useful system. The NPSNET project itself is a good study of the complexity of constructing 3D virtual worlds with available commercial technology and why fully interactive and detailed virtual worlds are not yet even on the horizon despite media promises. We are optimistic and hope that by "pushing the envelope" of real-time, workstation-based virtual reality, we are finding a way to reach the goal of a fully interactive and detailed virtual world.

Acknowledgments

We wish to acknowledge the sponsors of our efforts, in particular George Lukes of the USA Engineer Topographic Laboratories, Michael Tedeschi of the USA Test and Experimentation Command, John Maynard and Duane Gomez of the Naval Ocean Systems Center, San Diego, LTC Dennis Rochette, USA of the Headquarters Department of Army AI Center, Washington, D.C. and Carl Driskell of PM-TRADE.

References

1. Akely, Kurt, "The Hidden Charms of the Z-Buffer," *IRIS Universe*, Vol. 11, March 1990, pp. 33-34.

2. Clark, James, "Hierarchical Geometric Models for Visible Surface Algorithms," *CACM*, Vol. 19, No. 10, October 1976, pp. 547-554.

3. Fichten, Mark and Jennings, David, *Meaningful Real-Time Graphics Workstation Performance Measurements*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

4. Glassner, Andrew, editor, *An Introduction to Ray Tracing*, Academic Press, San Diego, CA, 1990, pp. 35-78

5. Institute for Simulation and Training, "Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation", Military Standard (DRAFT), IST-PD-90-2, Orlando, FL, September 1991.

6. Lang, Eric and Wever, Peters, *SDIS Version 3.0 User's Guide: Interchange Specification, Class Definitions, Application Programmer's Interface*, BBN Systems and Technologies, Bellevue, WA, August 1990.

7. Mackey, Randall, *NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation*, M.S. Thesis, Naval Postgraduate School, Monterey, California, September 1991.

8. Monahan, James, *NPSNET: Physically-Based Modeling Enhancements to an Object File Format*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1991.

9. Nizolak, Joseph Jr., Drummond, William T. Jr., and Zyda, Michael J. "FOST: Innovative Training for Tomorrow's Battlefield," Field Artillery, HQDA PB 6-90-1, February 1990, pp. 46-51.

10. Polcrack, Jane, *Using Solid Modeling Techniques to Construct Three-Dimensional Icons for a Visual Simulator*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1991.

11. Pope, Arthur, "The SIMNET Network and Protocols", BBN Report No. 7102, BBN Systems and Technologies, Cambridge, MA, July 1989.

12. Samet, Hanan, "Applications of Spatial Data Structures", Addison Wesley, 1990, p. 426.

13. Shaffer, Clifford, "Fast Circle-Rectangle Intersection", *Graphics Gems*, Ed. Andrew Glassner, Academic Press, Boston, 1990, pp. 51-53.

14. Silicon Graphics Computer Systems Inc., Graphics Library Reference Manual, C edition, IRIS-4D series, 1990

15. Silicon Graphics, Inc. "Network Communications," Document Version 1.0, Document Number 007-0810-010, Mountain View, CA, 1990.

16. Tanenbaum, Andrew, "Computer Networks", Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1989, pp. 146-148.

17. Thorpe, Jack "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting," *Proceedings of the Ninth Interservice Industry Training Systems Conference*, November 1987.

18. Zyda, Michael "3D Visual Simulation for Graphics Performance Characterization," NCGA '90 Conference Proceedings, Vol. I, 22 March 1990, pp. 705-714.

19. Zyda, Michael, Fichten, Mark, and Jennings, David H. "Meaningful Graphics Workstation Performance Measurements," Computers & Graphics, Vol. 14, No. 3, 1990, Great Britain: Pergamon Press, pp.519-526.

20. Zyda. Michael, McGhee, Robert, Kwak, S., Nordman, D.B., Rogers, R.C., and Marco, D. "3D Visualization of Mission Planning and Control for the NPS Autonomous Underwater Vehicle," IEEE Journal of Oceanic Engineering, Vol. 15, No. 3, July 1990, pp. 217-221.

21. Zyda, Michael, McGhee, Robert, McConkle, Corinne M., Nelson, Andrew H. and Ross, Ron S. "A Real-Time, Three-Dimensional Moving Platform Visualization Tool," Computers & Graphics, Vol. 14, No. 2, 1990, Great Britain: Pergamon Press, pp.321-333.

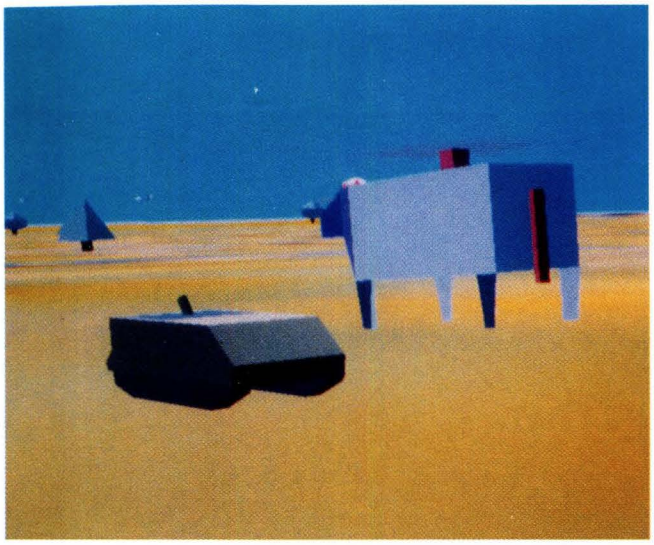
22. Zyda, Michael, McGhee, Robert, Ross, Ron, Smith, Doug and Streyle, Dale "Flight Simulators for Under $100,000," IEEE Computer Graphics & Applications, Vol. 8, No. 1, January 1988, pp. 19-27

23. Zyda, Michael and Pratt, David "3D Visual Simulation as Workstation Exhaustion," Proceedings of Ausgraph 90, Melbourne, Australia, 10 - 14 September 1990, pp. 313-328.

24. Zyda, Michael and Pratt, David "Zydaville," on ACM SIGGRAPH Video Review, Vol. 60, August 1990, entitled "HDTV & The Quest for Virtual Reality". The video segment shows our NPSNET system and a brief interview of Professor Zyda.

25. Zyda, Michael and Pratt, David "NPSNET: A 3D Visual Simulator for Virtual World Exploration and Experimentation, "1991 SID International Symposium Digest of Technical Papers, Volume XXII, 8 May 1991, pp. 361-364.

26. Zyda, Michael, Wilson, Kalin, Pratt, David, and Monahan, James, *NPSOFF: An Object Description Language for Supporting Virtual World Construction*, Naval Postgraduate School, Monterey, CA, October 1991, in preparation.



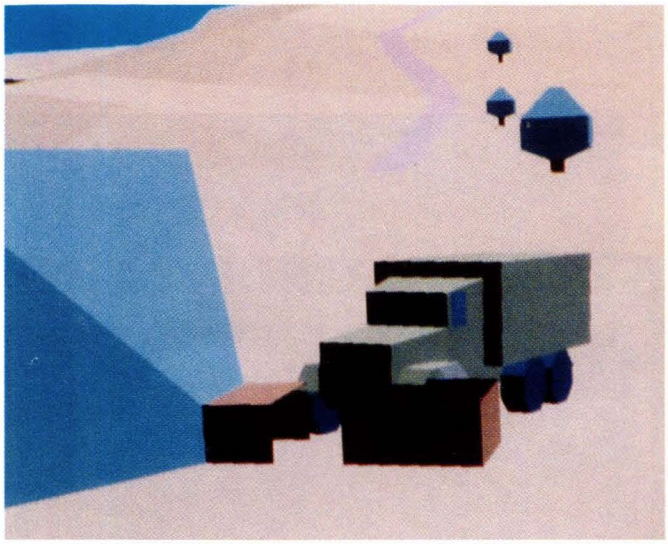


Plate 1: (upper left) Helo-cow stalking a M-106 Self-propelled mortar.

Plate 2: (upper right) Results of collision between a M-35 2 1/2 ton truck traveling at medium speed and a tree.

Plate 3: (middle left) The Helo-cow's round nearly impacts with an M-2FAADS tank.

Plate 4: (bottom left) Multiple formations of tanks and aircraft on tracks generated by NPSNET-MES. V-22 Ospreys and AH-1T Cobras provide close air support.

Zyda, Pratt, Monahan, and Wilson, "NPSNET: Constructing a 3D Virtual World"